Contents

Chapter 1

Introduction

1.1 GNU Radio

GNU Radio[1] is a free & open-source software development toolkit that provides signal processing blocks to implement software radios. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in hobbyist, academic and commercial environments to support both wireless communications research and real-world radio systems.

A software radio is a radio system which performs the required signal processing in software instead of using dedicated integrated circuits in hardware. The benefit is that since software can be easily replaced in the radio system, the same hardware can be used to create many kinds of radios for many different transmission standards; thus, one software radio can used for a variety of applications.

GNU Radio performs all the signal processing. You can use it to write applications to receive data out of digital streams or to push data into digital streams, which is then transmitted using hardware. GNU Radio has filters, channel codes, synchronization elements, equalizers, demodulators, vocoders, decoders, and many other elements (in the GNU Radio jargon, we call these elements blocks) which are typically found in radio systems. More importantly, it includes a method of connecting these blocks and then manages how data is passed from one block to another. Extending GNU Radio is also quite easy; if you find a specific block that is missing, you can quickly create and add it.

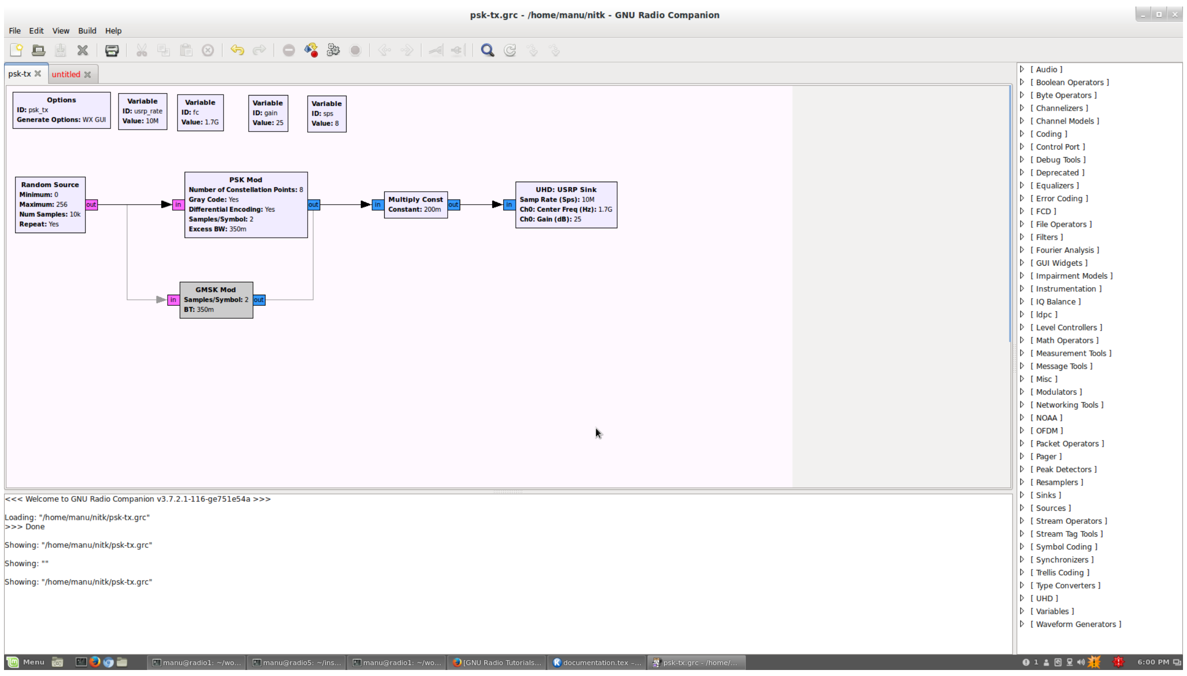
Since GNU Radio is software, it can only handle digital data. Usually, complex baseband samples are the input data type for receivers and the output data type for transmitters. Analog hardware is then used to shift the signal to the desired center frequency. That requirement aside, any data type can be passed from one block to another - be it bits, bytes, vectors, bursts or more complex data types.

GNU Radio applications are primarily written using the Python programming language, while the supplied, performance-critical signal processing path is implemented in C++ using processor floating point extensions, where available. GNU Radio Companion(GRC) is a Simulink-like graphical tool to design signal processing flow graphs.

GNU Radio supports several radio front-ends, either natively or through additional out-of-tree modules. We will be using Ettus Research USRP platform and RTL-SDR TV tuners.

1.2 First flow-graph

Lets make our first flow-graph. Our aim is to generate a sinusoid and play it on audio output. Open GNU Radio companion by executing gnuradio-companion on a terminal. It opens a window similar to the one given in Figure 1.1. When we open a new flow graph, a default graph as shown in Figure 1.2 comes up. The completed flow-graph is shown in Figure 1.3. When we execute this flow-graph, it is a python program that is getting executed.



Options
ID: psk_tx
Generate Options: WX GUI

Variable
ID: usrp_rate
Value: 10M

Variable
ID: fc
Value: 1.7G

Variable
ID: gain
Value: 25

Variable
ID: sps
Value: 8

Random Source
Minimum: 0
Maximum: 256
Num Samples: 10k
Repeat: Yes

PSK Mod
Number of Constellation Points: 8
Gray Code: Yes
Differential Encoding: Yes
Samples/Symbol: 2
Excess BW: 350m

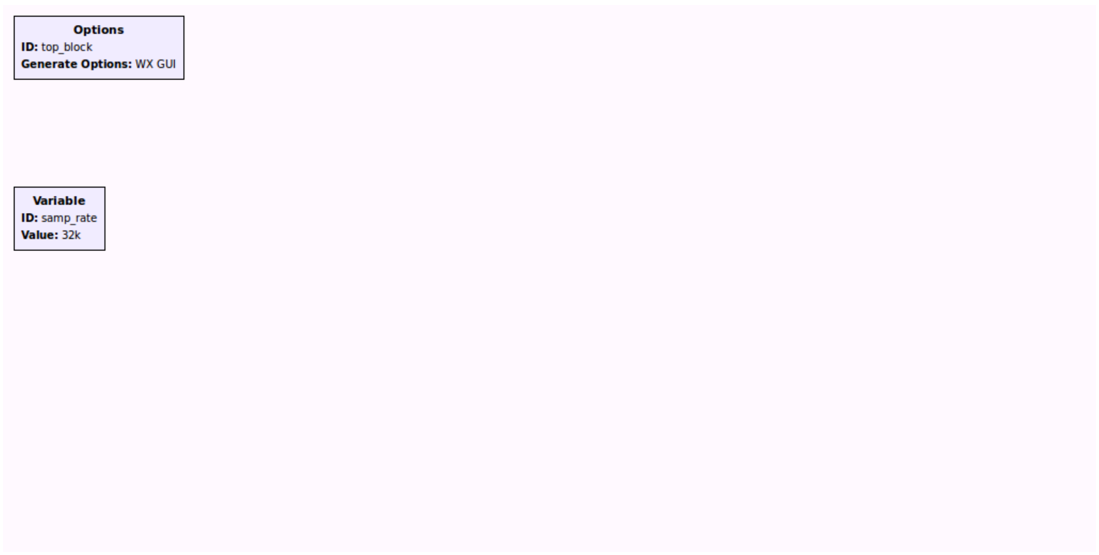
Multiply Const
Constant: 200m

UHD: USRP Sink
Samp Rate (Sps): 10M
Ch0: Center Freq (Hz): 1.7G
Ch0: Gain (dB): 25

GMSK Mod
Samples/Symbol: 2
BT: 350m

```
<<< Welcome to GNU Radio Companion v3.7.2.1-116-ge751e54a >>>

Loading: "/home/manu/nitk/psk-tx.grc"
>>> Done

Showing: "/home/manu/nitk/psk-tx.grc"

Showing: ""

Showing: "/home/manu/nitk/psk-tx.grc"
```

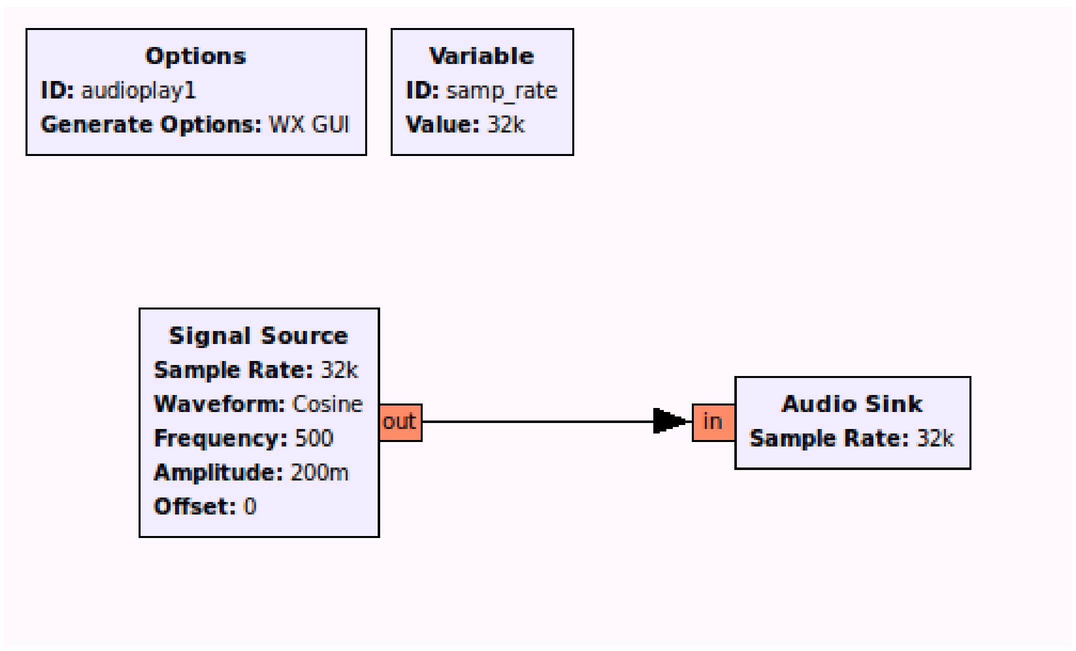
Figure 1.1: GNU Radio companion. The area containing the blocks in the center is the place where we assemble the flow-graphs. We can drag and drop different signal processing blocks to the flow-graph from the list-box on the right. The text-box on the bottom displays the output written to the terminal.



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 32k

Figure 1.2: Constructing a flow graph. On the left top we see the options block. The variable block defines a variable called samp_rate. We can change the value associated with this variable by double clicking on it and editing the property window that pops up. One can use this variable in other block by putting the variable id i.e, samp_rate.



ID: audioplay1
Generate Options: WX GUI

ID: samp_rate
Value: 32k

Sample Rate: 32k
Waveform: Cosine
Frequency: 500
Amplitude: 200m
Offset: 0

Sample Rate: 32k

Figure 1.3: Playing 500Hz sinusoid on audio output. Take signal source and audio sink block from the list of blocks. Set the data type of signal source to float. Set the frequency to 500Hz and sampling rate to 32k. Set the sampling rate in the audio sink block to 32k. Connect these blocks and execute the flow-graph.

Appendices

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: Audioplay1
# Generated: Wed Jun 25 12:46:12 2014
##################################################

from gnuradio import analog
from gnuradio import audio
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class audioplay1(grc_wxgui.top_block_gui):

    def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="Audioplay1")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 32000

        ##################################################
        # Blocks
        ##################################################
        self.audio_sink_0 = audio.sink(samp_rate, "", True)
        self.analog_sig_source_x_0 = analog.sig_source_f(samp_rate, analog.GR_COS_WAVE,
    500, 0.2, 0)

        ##################################################
        # Connections
        ##################################################
        self.connect((self.analog_sig_source_x_0, 0), (self.audio_sink_0, 0))


# QT sink close method reimplementation

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"
    parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
    (options, args) = parser.parse_args()
    tb = audioplay1()
    tb.Start(True)
    tb.Wait()
```

Listing 1: The python program corresponding to the flow-graph in Figure 1.3

Bibliography

[1] http://gnuradio.org

[2] http://www.linear.com/product/LTC5598

[3] http://www.linear.com/product/LTC6946